DEPARTMENT OF THE AIR FORCE

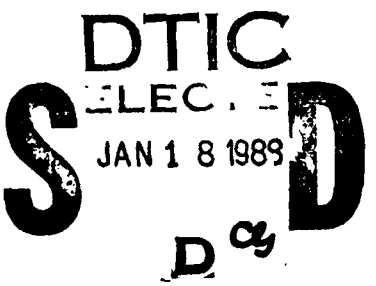
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

(1)

AFIT/GE/ENG/88D-43



IMPLEMENTATION OF THE NETOS OPERATING SYSTEM IN ADA WITH MODIFICATIONS TO ALLOW VARIABLE LENGTH MESSAGES

THESIS

Robert Rodriguez
Captain USAF

AFIT/GE/ENG/88D-43

Approved for public release; distribution unlimited

AFIT/GE/ENG/88D-43

IMPLEMENTATION OF THE NETOS OPERATING SYSTEM IN ADA WITH MODIFICATIONS TO ALLOW VARIABLE LENGTH MESSAGES

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

by

Robert Rodriguez

Captain USAF

December 1988

Approved for public release; distribution unlimited

Preface

The purpose of this thesis effort was to redesign the Network Operating System (NETOS) software in the programming language Ada with modifications to support variable length messages. This effort was prompted by a need to support software engineering courses and computer network courses at AFIT with a unique academic laboratory environment which encourages modern software engineering principles.

Software development was patterned after the software engineering development methodology presented in EENG 5.93, Software Engineering, which consists of requirements analysis, design, implementation, and testing. Each phase was completed; however, the data dictionaries for both the SADT diagrams and the structure charts were not developed due to time limitation.

I wish to express my appreciation to those who have supported and helped me in this thesis effort. I am indebted to Charles Powers and Dr. Hartrum for their assistance with the LSINET. I wish to thank my thesis advisor, Capt B. George, for his continual support and encouragement. I wish to thank my wife, Alicia, and our children, Jason and Jenny, for their patience and support throughout my tour at AFIT. And most importantly, I wish to thank my Lord and Savior, Jesus Christ, who gave me the strength to complete this work (Philippians 4:13).

Robert Rodriguez

Table of Contents

List of Figures

List of Tables

AFIT/GE/ENG/88D-43

Abstract

This thesis redeveloped the Network Operating System (NETOS) software, which is patterned after the OSI 7-Layer Model and runs on a Local Area Network at the Air Force Institute of Technology, from the programming language C to the programming language Ada, with modifications to support variable length messages.

The approach taken used a software development methodology which contains the following phases; requirements analysis, design, implementation, and testing. The requirement analysis phase consisted of an enumerated listing of the requirement specifications supported by SADT diagrams. The design phase transformed these diagrams to a structural chart representation of the design. Implementation of the software was written in Janus/Ada for the work stations and Whitesmith C for the central system. Testing was an integral part of the implementation phase and was accomplished at each level of the 7-Layer model.

IMPLEMENTATION OF THE NETWORK OPERATING SYSTEM IN ADA WITH MODIFICATIONS TO ALLOW VARIABLE LENGTH MESSAGES

I. Introduction

Background

LSINET is a local area network (LAN) located at the School of Engineering, Air Force Institute of Technology (AFIT), Wright-Patterson Air Force Base, Ohio. This network provides a unique academic laboratory environment in support of software engineering courses, computer network courses, and various research projects. The LSINET originally consisted of LSI-11/02 microcomputers, thus the name LSINET. Later the LSI-11/02s were upgraded to LSI-11/23s. And within the last year, the work stations have been replaced with Zenith Z-248 microcomputers. The central system and one station, however, still remain as LSI-11/23 microcomputers. The development effort of the LSINET was accomplished through the NETOS (Network Operating System) project.

> "The NETOS (Network Operating System) project was conceived to support a required graduate level sequence in computer systems which includes courses in software engineering, operating systems, and architecture followed by a keystone laboratory that would allow students to apply software engineering techniques and methodologies to a large scale development. It

was planned from the beginning with a dual purpose in mind. The first purpose was pedagogical: to define a project that would be large and complex enough to allow the realistic application of software engineering principles and yet be achievable in a single 10-week quarter. The second purpose was to provide improved support for the software systems laboratory course to make software development more convenient for the students. This laboratory course was conducted using about a half dozen LSI-11/02 microcomputers, each with dual 8-inch single-density disks and only a few with printers. This latter goal included improved printing support by network access to the printers from any microcomputer, centralized file storage to support multi-team programming efforts, and upload/download access between the lab's microcomputers and the school's VAX-11/780.

The original concept was simply to network the LSI-11/02s together in the simplest manner possible. This limited LAN was to become known as LSINET. The project was then to be the development of a network operating system which would provide a limited number of network services, including printer spooler, and file storage and retrieval. All software was developed in C except where assembly language was necessary. The initial design involved a layered, top-down approach involving **four** ad hoc layers. During the first laboratory effort, it became clear that the lower levels needed redesigned and better organized. The decision was then made to implement a version of the ISO seven layer protocol. This had not only the objective of supporting the computer systems sequence and laboratory, but also of providing a working ISO model to support our two-quarter graduate sequence in computer networks. Although the LSINET was certainly a small, limited LAN, and despite the fact that most documented operational networks tend to lump the upper three or four layers together, the decision was made to design and implement explicitely all seven layers"(Hartrum, 1988).

NETOS has since gone through several modifications and enhancements. Table I lists the chronological development of NETOS. The original concept was based on an ad hoc design of four layers. Several problems were encountered with this initial design. This led to a more structured and thought out design which was based on the International Standards Organization (ISO) Open Systems Interconnection (OSI) 7-layer model. All seven layers including a Data Base Management System (DBMS) as layer seven were implemented during the Spring Quarter 1984. During the Summer Quarter 1984, a gateway node was added to the network to allow interconnection to the Universal Network Interface Device (UNID). During the Fall Quarter 1984, an interrupt version of Layer 2 was designed to allow queueing of packets. This allowed network servers to receive new packets while continuing to service an already received packet. Also, the gateway software was modified to directly interface to a VAX-11/780 running VMS without the use of the UNID. The Spring Quarter 1985 was used to validate and test layers 1-3. And finally in the Winter Quarter 1988, the LSI-11/23 work stations were replaced with Z-248 microcomputers (Hartrum, 1988).

Table I.
NETOS Development Chronology
(Hartrum, 1988)

	- First non-ISO NETOS incl Spooler & MSS
Winter '84	- NETOS ISO SADT developed: Dated March 1984, revised 4/12/84
Spring '84	- Initial ISO Layers 2-7 incl DBMS DD stored on VAX using editor
Summer '84	- Initial Gateway to UNID
Fall '84	- Layer 2 interrupt; extended DBMS; VMS interface DD on VAX w/Ingres
Spring '85	- Layer 1; cleanup ISO 2 & 3; add UNIX, CP/M, MS-DOS
Winter '88	- Converted from LSI-11 to Z-248 Work Stations

The topology of the LSINET is shown in Figure 1. Upper-case letters are used to identify each microcomputer. This thesis effort is restricted to a portion of the network consisting of nine Zenith Z-248 microcomputers (Systems A and C-J) and one LSI-11/23 microcomputer (System B).

A crucial node in the network is the central system (System B). The central system is the center node of the star and is responsible for routing packets between satellite computers. Polling is used by the central system to detect if any other system desires to communicate with it.

All software was developed in the C programming

language and based on the ISO OSI seven layer model. The ISO OSI seven layer model is shown in Figure 2. The exception is the central system which provides network management functions residing just above Layer 2 of the ISO model. Figure 3 shows the relationship of the central system's protocols layers in contrast to the rest of the network.

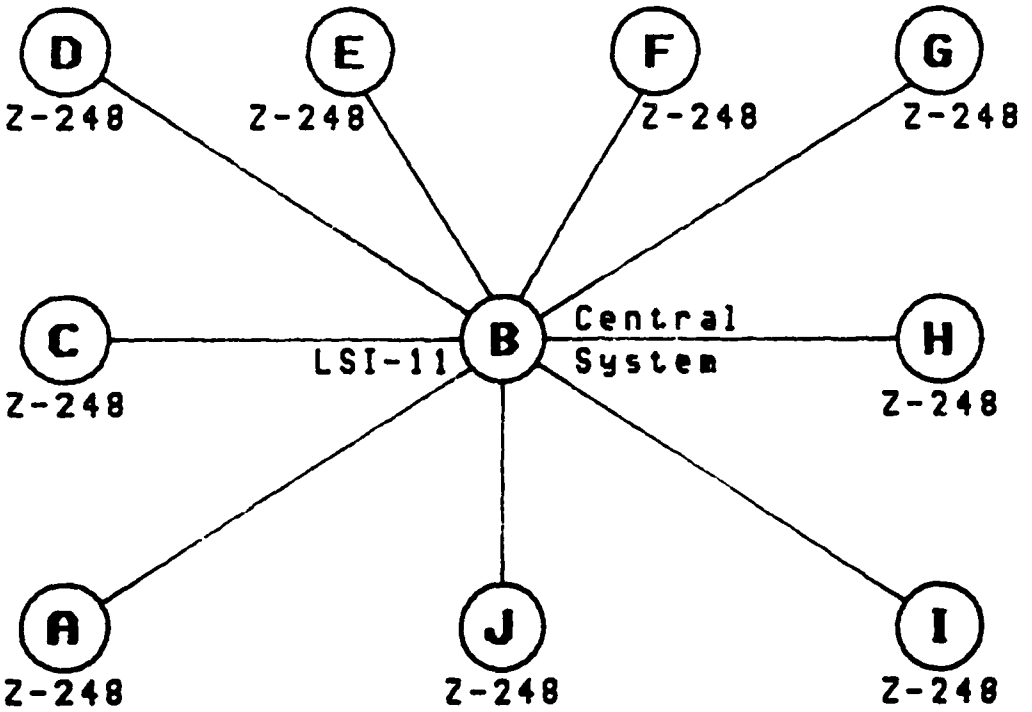


Figure 1. LSINET TOPOLOGY

LEVEL 7
Application Layer

LEVEL 6
Presentation Layer

LEVEL 5
Session Layer

LEVEL 4
Transport Layer

LEVEL 3
Network Layer

LEVEL 2
Data Link Layer

LEVEL 1
Physical Layer

Figure 2. ISO 7-layer Protocol Model

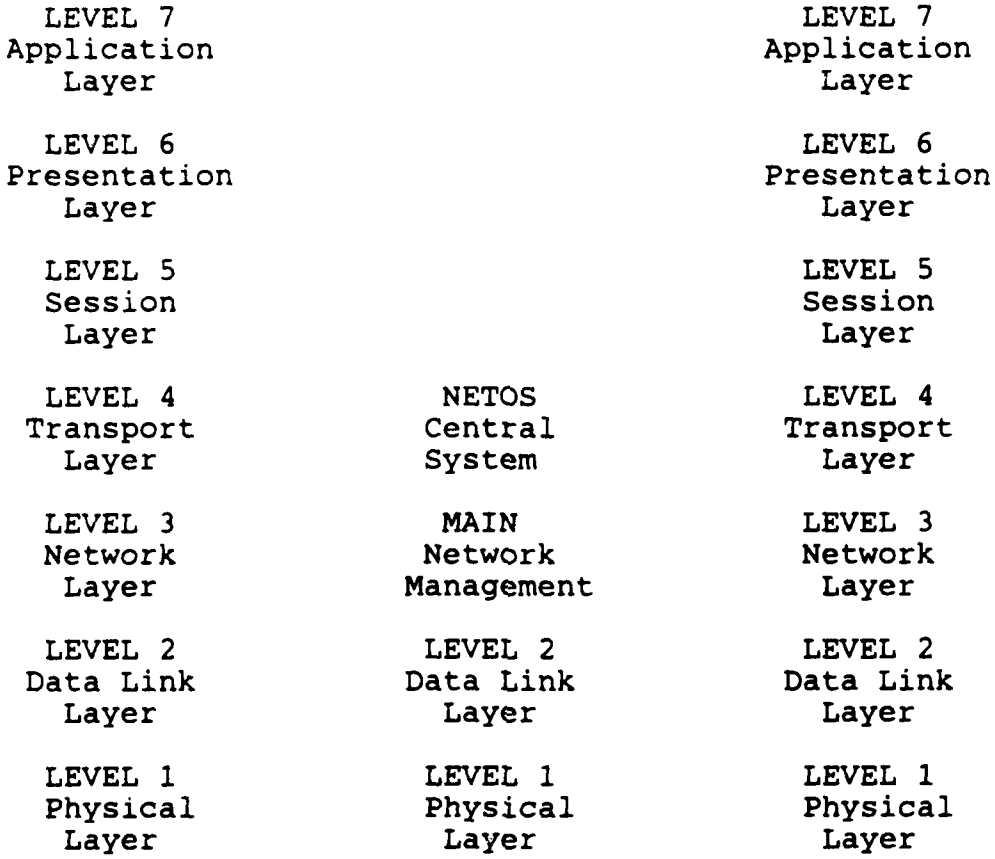


Figure 3. Central System Protocol Model
(Hartrum, 1988)

Problem

The major problem appears to center around the lack of cohesion and communication in the development of the LSINET. The evolving development spanned several academic quarters. The projects for each quarter were subdivided into separate student groups. Furthermore, the network not only supported software engineering development but also various other activities such as; computer network courses, student thesis, student independent special study, and faculty research. Due to this lack of cohesion and communication, the LSINET contains numerous discrepancies and shortcomings which hampers its ability to provide a useful and productive system for an academic and research environment.

After some hands-on-experience with the operation of the LSINET and examination of the source code, several discrepancies were encountered. Unfortunately, only the lower three layers of the network were in operation on the Z-248s. Therefore, all problem areas discovered were limited to these three layers. Two discrepancies encountered are described as follows.

In the Test ISO Layer 2, send2 module, the error codes which determine the appropriate error message to be displayed do not coincide with the error codes returned from the ISO Layer 2, send_packet module. Table II lists the error codes found in both modules.

Table II.
Send2 and send_packet error codes

Error	send2 error code	send_packet error code
No error	0	0
Timeout waiting for TA	2	202
Timeout waiting for ACK/NAK	3	204
Received a NAK	6	205
Unrecognized error	all others	-
Timeout sending TR	-	201
Non-TA received	-	203
Non-ACK/NAK received	-	206

Also, it was observed that the error message displayed on the central system's monitor are not always cleared. This makes detection of errors confusing, since the operator is unable to determine if the error belonged to the current message or the previous message.

A second problem resides in the language in which the present NETOS software is written. The current software is written in the language C. However,

> "As early as 1975, a few far-sighted individuals in the United States Department of Defense recognized the impending software challenge and realized that an effective software development system should be founded no a standard programming language with features that would encourage modern programming practices. They defined the requirements, recognized that no existing language would support such requirements, and hence commissioned the worldwide language design competition that produced Ada" (Booch, 1986).

Since Ada has been chosen to be the standard high level language as mandated by the DOD, AFIT has begin instructing all software design efforts utilizing Ada wherever possible. So, in order to allow students academic experience in coding in Ada; the NETOS software, which supports courses in software engineering and computer networks, must be coded in the programming language Ada.

Another problem is that the present system supposably allows for variable packets sizes to be transmitted. However, to do so requires that the source, destination, and central system all be set to the same packet size. In other words, each node that processes the packet must be configured to the same packet size. The major problem here is that to change the packet size of the central system requires the operator to manually interrupt the central node. And, the configuration of the central system dictates the configuration of the rest of the network since all traffic passes through the central node. Thus, the system at present does not allow for true variable length packets. The system should be modified to include as Dr. Hartrum states, "The addition of the ability for Layer Two to handle true variable length packets, where the packet size is carried along in the frame, would not only allow dynamic testing of various packet sizes, but would also allow for application programs to use a packet size best suited to the application" (Hartrum, 1988).

Scope

The scope of this thesis effort was to correct discrepancies, make enhancements, and implement the existing LSINET in Ada. Due to the size and complexity of the LSINET, this effort was restricted to the Central System and the Work Stations: A, C, D, E, F, G, H, I, and J. In order to stick with the original concept and provide a simple network, one that is easily understood and has a sound design which can be easily upgraded, this effort limited itself to the non-interrupt receive message version. Furthermore, since the interrupt version is primarily used by the network servers and since the network servers have yet to be implemented in Ada, the interrupt version is not useful at this time. However, the non-interrupt version is required. Unfortunately, since the Central System was not replaced with a Z-248 but still remains a LSI-11, the software written for the Central System remained in C as the RT-11 operating system do not support Ada.

This thesis effort was limited to the ISO Layers 1-6. Layer 7 software was not implemented. All application programs (Layer 7) must be supplied by the user.

This thesis effort did not attempt to make any changes to hardware, including reconfiguring the LSINET, other than utilizing Comm Port #2 of the Work Stations rather than Comm Port #1. This freed up Comm Port #1 to be used for other purposes such as a modem or mouse.

Assumptions

It is assumed that simplicity and understandability are of prime importance in the development of the software, since the network's purpose is to provide an academic laboratory environment. Productivity issues such as throughput and turnaround time are assumed to be of lesser importance.

Although the old network was extremely reliable, it is assumed that the transmission across the network is unreliable. This assumption is made to allow portability of the software to a less reliable environment.

Standards

The LSINET was developed based on the ISO OSI 7-layer network model. This model is a highly structured hierarchy of protocols organized as a series of layers. Each layer provides service to the next higher layer above it while shielding the higher layer from the implementation details of the lower layers. All layers, except the lowest layer, carry on a virtual conversation with its corresponding layer on another machine. However, the actual communication is to the layer below it. Only at the lowest layer is there a physical communication to another machine. The layers, the protocols associated with each layer, and the interface between layers are shown in Figure 4. Each layer will be described, starting with the top layer (Zimmerman, 1980).

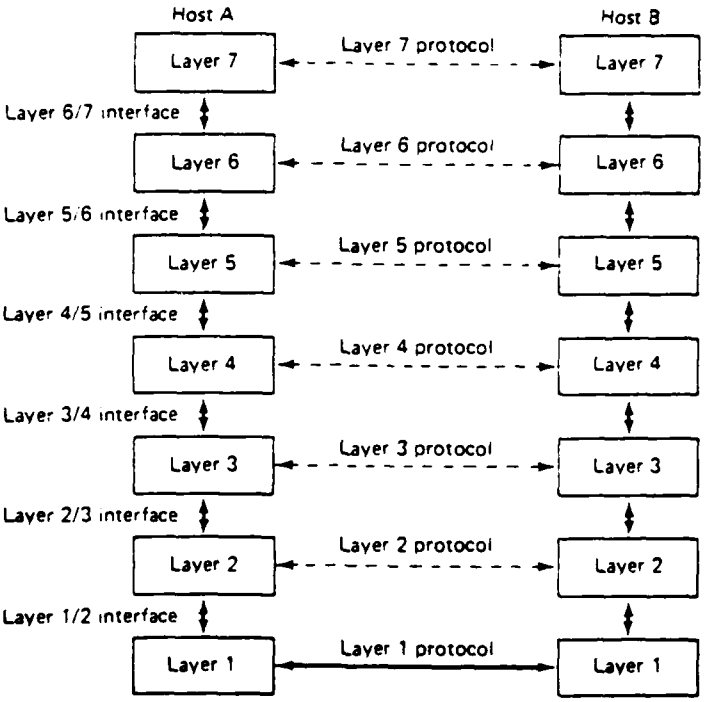


Figure 4. Layers, protocols, and interfaces.
(Tanenbaum, 1981: Fig 1-5)

Layer 7 - The Application Layer.

The application layer consists of the various user application programs. The details of the protocol, which establishes the rules and guidelines for communication between two application programs, are left up to the users.

Layer 6 - The Presentation Layer.

The presentation layer provides transformation services. Typical transformation services provided by layer six are: data compression, encryption, conversion between character codes (e.g. ASCII to EBCDIC), and various format conversion to compensate for incompatible terminals.

Layer 5 - The Session Layer.

Since the presentation layer merely provides transformation services, the session layer is the first layer that interfaces the user to the network. This layer is responsible for establishing a connection between two users. This connection is called a session. Once a session has been established, layer five must then maintain the session. Finally, layer five must be able to terminate a session.

Layer 4 - The Transport Layer.

The transport layer is responsible for providing an end-to-end transport service between processes. "It maps multiple processes at the Session layer level into a single

network communication link by establishing and deleting connections on behalf of Layer 5 and keeping track of which connection is associated with which host process" (Hartrum, 1988). Layer four also establishes messages as its units of information.

Layer 3 - The Network Layer.

The network layer is primarily concerned with the routing of packets through the network. Packets, which are formed from the messages obtained from layer four, are the units of information communicated in layer three. Based on the destination, this layer sends the packets out over the best available route.

Layer 2 - The Data Link Layer.

The data link layer is responsible for providing error free transmission of data frames over a link. Frames, which are formed from the packets obtained from layer three, are the units of information communicated in layer two. The frames typically include an error detection code. This is used at the receiving end to determine if the transmission was error free. Based on this information, the receiver can then notify the sender whether the transmission was successful or not. If not successful, the sender can then retry transmitting the frame again.

Layer 1 - The Physical Layer.

The physical layer is the only layer that actually sends data from one machine to another. Its purpose is to transmit and receive raw bits over a communication medium. Primary concerns involve the mechanical, electrical, and procedural characteristics of the communication medium (Tanenbaum, 1981).

Materials and Equipment

The materials and equipment which were required for this thesis effort were all available on the LSINET. The computer systems involved consisted of the Central System (LSI-11/23) and the Work Stations; A, C, D, E, F, G, H, I, and J (all Z-248s). A validated JANUS Ada compiler had recently been obtained by AFIT and was used to implement Ada on the Work Stations.

Approach

The approach taken in this thesis effort was patterned after the software development methodology presented in EENG 5.93, Software Engineering, and EENG 6.90, Software Systems Laboratory. The approach included requirements analysis, design, implementation, and testing.

The requirements analysis primarily consisted of an enumerated list of requirements specifications supported by SADT diagrams. This phase relied heavily on earlier work as the basic requirements did not change very much. The goal was to determine all requirements necessary to provide an

enhanced LSINET capable of supporting academic and research work at the Air Force Institute of Technology.

The design phase followed a top-down design approach. An object oriented design approach was considered, but much of the software engineering principles, such as information hiding, abstraction, and modularity, inherent in an object oriented design are also inherent in the ISO 7-Layer Model. Also the hierarchy structure of the ISO 7-Layer Model lends itself well towards a top-down design approach. The design phase was based on structure charts derived from the SADT diagrams produced in the requirements analysis. Transform and Transaction analysis was to derive the structure charts. Implementation consisted of coded modules. Coding followed a bottom-up approach, beginning with Layer 1 and working up to Layer 6. This approach was chosen to allow the layers to be built upon already existing and tested layers. This reduced the amount of stub and dummy modules required during testing. Testing was an integrate part of the implementation phase. Test modules were specifically designed for each module. And, each layer was thoroughly tested before work on the next layer was begun.

Presentation Sequence

The second chapter consists of the requirements analysis of the network. Requirements are subdivided into their appropriate layer of the seven layer model. SADT

diagrams also are provided.

The third chapter describes the design, coding, and testing effort. Included in this chapter are all design considerations supported by structure diagrams. The coding effort is described for each layer. The test approach is explained and is supported by a test plan and the results of the testing.

The last chapter summarizes this thesis effort. This includes conclusions and recommended future follow-on efforts.

II. Requirements Analysis

Introduction

The requirements analysis was based primarily upon the current NETOS system. Unfortunately, there does not exist a document containing an enumerated listing of requirements specifications for the current system. Therefore, the requirements analysis for the new system were derived from the observation of the current system, from a document containing a narrative form of the functional specifications of the current system (HARTRUM:88), and from new requirements developed in this thesis effort. The major portion of these new requirements are related to requiring the new NETOS system to handle variable length frames, packets, and messages.

This chapter will be presented in the following manner. First, the user's applications program (Layer 7) functional specifications will be described in a narrative format. Although Layer 7 is not part of this thesis effort, this information is provided in order to establish a standard basis for the interface between Layer 7 and Layer 6. Next, the central system's functional specification will be described in a narrative format followed by an enumerated listing of the central system's requirements specifications. And finally, the ISO Layers 1-6 will described in the like

manner as the central system. SADT diagrams of the central system and Layers 1-6 are kept on file by the LSINET manager, presently Dr. Hartrum.

Layer 7

Although Layer 7 is not part of this thesis effort, this information is provided to give some understanding of the interface required by Layers 7 and 6. Layer 7 consists of any application programs that makes subroutine calls to Layer 6 in order to access the network. These application programs can provide such network services as a Printer Spooler, a Mass Storage System, or a Data Base Management System. Each network service consists of two corresponding programs, the server program and the user access program.

Before a network service can be provided, both the server program and the user access program must initialize Layer 6. Then, the initiating program must send a request for service message to the corresponding server program. This is required in order to determine if the server program is online, ready, and capable of handling the request. Therefore, this request message must contain all necessary information to allow the service program to reach a decision as to accept the request or reject it.

The server program must be capable of sending a status message back to initiator. The status message indicates whether the request is accepted or rejected; and if

rejected, the reason why. If accepted, the status message may contain further information for the initiator.

Once the appropriate handshaking of requesting a service and receiving an accept status message is complete, the initiator and network server both must be capable of transmitting and receiving messages back and forth as required.

And finally, the initiator must be capable of sending a termination message when the session is complete.

Typically the initiator will be a user access program. However, there are special cases where the initiator will be a network server which is requesting service of another network server. For example, to print a list of files from the Mass Storage System, this server must request service of the Printer Spooler System (HARTRUM, 1988).

Central System

Narrative Functional Specifications.

The central system is the center node of a star topology network. Along with some limited network management functions, its primary function is to forward packets from the source to the destination and as such requires the use of a main routine and only Layers 1 and 2 of the ISO model.

Although the management functions are limited, they do provide the operator with some degree of management of the

NETOS system. The operator has the ability to select which ports are to be active. He/she has the ability to establish a work station as a monitor station. All traffic through the central system will be forwarded to each monitor station, which will allow the operator to examine the traffic. The central system will display to the operator the port being polled, the port in which a packet is being received, and the port through which the packet is being forwarded to its destination. And finally, the central system will display to the operator all errors encountered during transmission and reception.

The main routine software should initialize the central system. This should include building a port table from data loaded from a storage device, calling Layer 1 to initialize the ports, prompting the operator for port status (active or not active), prompting the operator in order to establish a work station which will monitor all network traffic, prompting the operator to exit to the operating system, and prompting the operator to activate the network.

Once the network is in operation, it should poll the ports for a transmit request. Once a TR has been received, it should respond with a TA. The receiving and transmitting of packets should follow the same procedures as described in the ISO Layers 1 and 2. However, instead of extracting the packet from the frame and passing it to Layer 3, the central

system software should forward the frame to its destination and any work stations defined to be network monitors.

Since there is no Layer 3, the central system software should interface with the user by sending information to the display. Information displayed should include all error messages and several status items such as source host of the packet being received, destination host to where the packet is being forwarded, and port being polled.

Enumerated Requirements Specifications.

SPEC_REQ_NUM_C.1 - Must be able to allow the user to exit to the operating system.

SPEC_REQ_NUM_C.2 - Must be able to allow the user to initialize the central system.

SPEC_REQ_NUM_C.2.1 - Must be able to load from a storage device any information required by the Central System initialization which is subject to change. This is to facilitate changes, by requiring changes only to the information in the storage device and not to the program's source code.

SPEC_REQ_NUM_C.2.2 - Must be able to prompt the user for port status. This allows the user to activate or deactivate a particular port.

SPEC_REQ_NUM_C.2.3 - Must be able to prompt the user for monitor status. This allows the user to establish or disestablish a particular work station to serve as a monitor of all traffic on the network. (All traffic should be forwarded not only to its destination but also to any work station established as a monitor station by the user.)

SPEC_REQ_NUM_C.3 - Must be able to poll all ports, seeing if there are any transmit requests.

SPEC_REQ_NUM_C.4 - Must be able to receive a variable length packet. See SPEC_REQ_NUM_2.3

SPEC_REQ_NUM_C.5 - If a valid packet was received, must be able to transmit the received packet to its destination and all active monitor stations. See SPEC_REQ_NUM_2.2

SPEC_REQ_NUM_C.6 - Must be able to display the status codes of the receptions and transmissions.

SPEC_REQ_NUM_C.7 - Must be able to display the port being polled, the port that is receiving the packet, and the port that is sending the packet.

SPEC_REQ_NUM_C.8 - Must return to the polling state upon any errors, after timeouts have expired, or upon a successful forwarding of a packet.

Layer 1

Narrative Functional Specifications.

Layer 1 provides the capability of initializing the RS-232 serial ports to a desired configuration. It also allows testing of the port to determine if a byte of data has arrived or if the port is ready to accept data for output. If the port is ready, Layer 1 provides the capability to transmit and receive a byte of data through a RS-232 serial port.

Enumerated Requirements Specifications.

SPEC_REQ_NUM_1.1 - Must be capable of initializing a RS-232 serial port.

SPEC_REQ_NUM_1.1.1 - Although the only ports currently available on the remote systems (Z-248s) capable of supporting the NETOS system are ports 1, 2 and 3; Layer 1 should have the capability of supporting port numbers greater than this.

SPEC_REQ_NUM_1.1.2 - Should allow baud rates of 1200, 2400, 4800, and 9600 bits per second.

SPEC_REQ_NUM_1.1.3 - The initialization parameters should be set for; no parity, one (1) stop bit, and eight (8) character bits.

SPEC_REQ_NUM_1.1.4 - Should return an error code if the baud input is not a valid selection.

SPEC_REQ_NUM_1.1.5 - Should return an error code if initializing the port failed.

SPEC_REQ_NUM_1.1.6 - Should return a status code indicating that the port was initialized successfully.

SPEC_REQ_NUM_1.2 - Must be capable of determining if a RS-232 serial port has data available to be read.

SPEC_REQ_NUM_1.2.1 - Should return a status code indicating if the port has data available to be read or not.

SPEC_REQ_NUM_1.3 - Must be capable of determining if a RS-232 serial port is available to write data to.

SPEC_REQ_NUM_1.3.1 - Should return a status code indicating if it is clear to write data to the port or not.

SPEC_REQ_NUM_1.4 - Must be capable of reading data from a RS-232 serial port.

SPEC_REQ_NUM_1.4.1 - Should be capable of reading a full byte of data, eight (8) bits, and not just the standard ASCII characters which utilize only seven (7) bits.

SPEC_REQ_NUM_1.5 - Must be capable of writing data to a RS-232 serial port.

SPEC_REQ_NUM_1.5.1 - Should be capable of writing a full byte of data, eight (8) bits, and not just the standard ASCII characters which utilize only seven (7) bits.

Layer 2

Narrative Functional Specifications.

Layer 2 provides the services of transmitting and receiving frames of variable length.

In order to transmit a frame, a frame is built from a packet of psize obtained from Layer 3. A checksum is calculated and included in the frame to provide an error detection capability. Also included in the frame is the frame length which is required due to the variable length of the frames. Next, the proper NETOS handshaking requirement must be observed. A transmit request (TR) must be sent to the central system which then responds by sending back a transmit acknowledge (TA). Layer 2 should allow several attempts in establishing the NETOS handshake requirement. Once that is established, the frame can be transmitted. The central system should respond with an acknowledgment (ACK) or nonacknowledgment (NAK), dependent on errors in the transmission. Layer 2 should allow several attempts to

transmit a frame. Based upon the response, Layer 2 will either relinquish control back to Layer 3, timeout, or try transmitting the frame again.

In order to receive a frame, the NETOS handshaking requirement again must be observed by waiting for a transmit request from the central system and responding with a transmit acknowledgment. Then, a frame can be received. Once the frame is received, the checksum needs to be validated. Depending on the results, an acknowledgment or nonacknowledgment will be sent back to the central system. Finally, if acknowledged, the packet will be extracted from the frame and sent to Layer 3.

Enumerated Requirements Specifications.

SPEC_REQ_NUM_2.1 - Layer 2 must be able to initialize all requirements necessary to operate at this level.

SPEC_REQ_NUM_2.1.1 - Must be able to load from a storage device any information required by Layer 2 initialization which is subject to change. This is to facilitate changes, by requiring changes only to the information in the storage device and not to the program's source code.

SPEC_REQ_NUM_2.2 - Must be capable of transmitting a variable length frame.

SPEC_REQ_NUM_2.2.1 - Must be capable of calculating a checksum from a variable length packet.

SPEC_REQ_NUM_2.2.2 - Must be capable of building a variable length frame from a variable length packet.

SPEC_REQ_NUM_2.2.2.1 - The frame should contain at least; a SOH, the frame length, the packet, the checksum, and a ETX.

SPEC_REQ_NUM_2.2.3 - When sending a frame, must be able to handle the NETOS handshaking requirement of sending a transmit request (TR) and waiting to receive a transmit acknowledge (TA).

SPEC_REQ_NUM_2.2.3.1 - Should allow several attempts in trying to establish the NETOS handshake.

SPEC_REQ_NUM_2.2.3.2 - Should return an error code if after a reasonable amount of time Layer 2 is unable to transmit a TR.

SPEC_REQ_NUM_2.2.3.3 - Should return an error code if after a reasonable amount of time Layer 2 has not received a TA.

SPEC_REQ_NUM_2.2.3.4 - Should return an error code if anything other than a TA is received when expecting a TA.

SPEC_REQ_NUM_2.2.3.5 - Should return a status code indicating that the NETOS handshake was accomplished successfully.

SPEC_REQ_NUM_2.2.4 - Should allow several attempts in trying to send a frame.

SPEC_REQ_NUM_2.2.5 - Must be able to receive an ACK/NAK reply.

SPEC_REQ_NUM_2.2.5.1 - Should return an error code if after a reasonable amount of time an ACK/NAK has not been received.

SPEC_REQ_NUM_2.2.5.2 - Should return an error code if anything other than an ACK or NAK is received.

SPEC_REQ_NUM_2.2.5.3 - Should return an error code if a NAK is received.

SPEC_REQ_NUM_2.2.5.4 - Should return a status code indicating that an ACK was received.

SPEC_REQ_NUM_2.3 - Must be capable of receiving a variable length frame.

SPEC_REQ_NUM_2.3.1 - When receiving a frame, must be able to handle the NETOS handshaking requirement of waiting for a transmit request (TR) and then sending a transmit acknowledge (TA).

SPEC_REQ_NUM_2.3.1.1 - Should return an error code if after a reasonable amount of time Layer 2 has not received a TR.

SPEC_REQ_NUM_2.3.1.2 - Should return an error code if Layer 2 has received something other than a TR.

SPEC_REQ_NUM_2.3.1.3 - Should return an error code if after a reasonable amount time Layer 2 is unable to transmit a TA.

SPEC_REQ_NUM_2.3.1.4 - Should return a status code indicating that the NETOS handshake was accomplished successfully.

SPEC_REQ_NUM_2.3.2 - Should return an error code if the first item of the frame is not a SOH.

SPEC_REQ_NUM_2.3.3 - Should return an error code if after a reasonable amount of time the entire frame has not yet been received.

SPEC_REQ_NUM_2.3.4 - Should be able to calculate and compare the packet checksum with the checksum sent in the packet.

SPEC_REQ_NUM_2.3.5 - Must be able to transmit an ACK/NAK reply based on the results of the checksum comparison.

SPEC_REQ_NUM_2.3.6 - Must be able to extract the packet from the frame received.

Layer 3

Narrative Functional Specifications.

Layer 3 is responsible for transmitting and receiving packets of variable length.

In order to transmit a packet, the packet first must be built from a message of msize received from Layer 4. Next, the destination host (Dst) code must be mapped to the proper I/O port in which to route the packet. Then, the packet can be sent to the destination process.

In order to receive a packet, the Src code is mapped to the proper I/O port in which the packet will be routed. Next, the packet is received from Layer 2. Then, the Src and Dst codes are evaluated to determine if the packet should be accepted or rejected. Last, if the packet is accepted the message is extracted from the packet and sent to Layer 4.

Layer 3 should also allow for the transmission and reception of system messages to be handled at the Layer 3 level.

Enumerated Requirements Specifications.

SPEC_REQ_NUM_3.1 - Layer 3 must be able to initialize all requirements necessary to operate at this level.

SPEC_REQ_NUM_3.1.1 - Must be able to load from a storage device any information required by Layer 3 subject to change. This is to facilitate changes, by requiring changes only to the information in the storage device and not to the program's source code.

SPEC_REQ_NUM_3.2 - Must be able to transmit a variable length packet.

SPEC_REQ_NUM_3.2.1 - Must be able to send a variable length Layer 3 System Packet. This packet will contain information for use at the Layer 3 level only.

SPEC_REQ_NUM_3.2.2 - Must be able to build a variable length packet from a variable length message.

SPEC_REQ_NUM_3.2.2.1 - All packets should contain a time stamp.

SPEC_REQ_NUM_3.2.3 - The proper port must be determined in which to route the packet.

SPEC_REQ_NUM_3.3 - Must be able to receive a variable length packet.

SPEC_REQ_NUM_3.3.1 - Must be able to receive a variable length Layer 3 System Packet.

SPEC_REQ_NUM_3.3.2. - Must be able to receive a particular packet (from a particular source host/process).

SPEC_REQ_NUM_3.3.2.1 - The proper port must be determined in which to receive the packet.

SPEC_REQ_NUM_3.3.3 - Must also be able to receive a packet from any source host/process.

SPEC_REQ_NUM_3.3.3.1 - All ports should be polled until a packet is received.

SPEC_REQ_NUM_3.3.4 - Must be able to extract the message from the packet.

SPEC_REQ_NUM_3.3.5 - Must be able to accommodate Layer 3 System Packets. (These are packets which are processed at Layer 3 and not passed up to the next Layer. Presently there are no requirements for Layer 3 System Packets, but future changes might include them to allow altering the routing table.)

Layer 4

Narrative Functional Specifications.

Layer 4 provides for the establishing of a logical connection between two host processes, maintaining the connection, and upon completion terminating the connection.

In order to establish a logical connection, Layer 4 must be able to to establish a channel number. This includes determining if a channel number is available, and if so, assigning that channel number to a source-destination transport address combination. The transport address consist of a physical host designator and a process number. After a channel has been assigned, a channel request message is built and sent to the destination Layer 4. The destination checks to see if the channel request message is a valid request for a channel. If valid, the destination establishes its own channel number. Depending on the results, the destination Layer 4 builds either a invalid response message or a valid response message. The reply is then sent back. After receiving the reply message, it is evaluated to determine if the connection was successful or

not. If successful, the channel number is returned to Layer 5. If not successful, the channel number is freed and a error message is returned.

Once a connection has been established, Layer 4 is able to send and receive messages. To send a message, the channel number is used to look up the corresponding source and destination transport addresses. If the data length to be transmitted is smaller than the variable message size specified by the application program, then the variable message size should be adjusted to the data size. If, on the other hand, the data length to be transmitted is larger than the variable message size, then the data should be broken up into lengths of message size and each portion transmitted in turn. When the end of data has been reached a special message should be transmitted signaling the end of data.

To receive a message, the channel number is used to determine the corresponding source-destination transport address. The original data that was sent is received by building the data from the collection of received subparts until the special message signaling the end of data is received.

Enumerated Requirements Specifications.

SPEC_REQ_NUM_4.1 - Layer 4 must be able to initialize all requirements necessary to operate at this level.

SPEC_REQ_NUM_4.1.1 - Must be able to load from a storage device any information required by Layer 4 subject to change. This is to facilitate changes, by requiring changes only to the information in the storage device and not to the program's source code.

SPEC_REQ_NUM_4.2 - When sending a transmission, first must be able to establish a connection.

SPEC_REQ_NUM_4.2.1 - Must assign an available channel for each connection.

SPEC_REQ_NUM_4.2.1.1 - Should allow for several different channels operating concurrently.

SPEC_REQ_NUM_4.2.1.2 - If attempting to establish a channel and all channels are already in use, should return an error code.

SPEC_REQ_NUM_4.2.1.3 - If attempting to establish a channel and successful, should return a status code indicating so.

SPEC_REQ_NUM_4.2.2 - Must be able to send a connect request message to the destination signaling it to also establish a connection.

SPEC_REQ_NUM_4.2.3 - Must be able to receive the destination's reply to the connect request message.

SPEC_REQ_NUM_4.2.4 - Must be able to decode the reply message.

SPEC_REQ_NUM_4.2.4.1 - If an invalid reply, should return an error code.

SPEC_REQ_NUM_4.2.4.2 - If a valid reply, should return a status code indicating so.

SPEC_REQ_NUM_4.3 - When receiving a transmission, first must be able to establish a connection.

SPEC_REQ_NUM_4.3.1 - Must be able to receive a connect request message from the source which is attempting to establish the connection.

SPEC_REQ_NUM_4.3.2 - Must be able to decode the connect request message.

SPEC_REQ_NUM_4.3.2.1 - If an invalid connect request message, should return an error code and send back an invalid connect request reply message.

SPEC_REQ_NUM_4.3.2.2 - If a valid connect request message and a channel can be assigned, should send back a valid connect request reply message.

SPEC_REQ_NUM_4.3.3 - Must assign an available channel for each connection.

SPEC_REQ_NUM_4.3.3.1 - Should allow for several different channels operating concurrently.

SPEC_REQ_NUM_4.3.3.2 - If attempting to establish a channel and all channels are already in use, should return an error code.

SPEC_REQ_NUM_4.3.3.3 - If attempting to establish a channel and successful, should return a status code indicating so.

SPEC_REQ_NUM_4.4 - Must be able to close a connection.

SPEC_REQ_NUM_4.4.1 - Should return an error code if an error is encountered in closing the connection.

SPEC_REQ_NUM_4.5 - Must be able to transmit variable length messages over a previously established connection.

SPEC_REQ_NUM_4.5.1 - Should transmit the message MSIZE bytes at a time.

SPEC_REQ_NUM_4.5.2 - If the message is less than MSIZE bytes, transmit only the required number of bytes in order to send the message.

SPEC_REQ_NUM_4.5.3 - Should some how indicate to the receiving end that the entire message has been transmitted.

SPEC_REQ_NUM_4.6 - Must be able to receive variable length messages over a previously established connection.

SPEC_REQ_NUM_4.6.1 - Should be able to determine if the message received is assigned to the particular channel.

SPEC_REQ_NUM_4.6.2 - Should be able to determine if the EOM has been reached.

SPEC_REQ_NUM_4.6.2.1 - If the EOM has not been reached, continue to build the submessages into the original message.

Layer 5

Narrative Functional Specifications.

Layer 5 is responsible for opening, maintaining, and closing a session between two processes. Layer 5 also maps the logical resource names to the transport address. A determination is made to see if the source and destination logical resource names are valid; and if valid, convert them to a transport address consisting of a physical host designator and a process number.

A logical connection must be established between the source and destination process before transmission can begin. Once a connection has been established, all subsequent transmission over the logical connection must

specify the connection number.

Message transmission and reception is accomplished by sending and receiving messages over a previously established connection.

File transmission is accomplished by reading a block of data at a time from a file and then transmitting it. The blocks of data are continually sent until the end of file is reached at which time a special block is sent indicating the end of data.

File reception is accomplished by receiving the blocks of data and writing them to a file until the special block is received.

Lastly, layer 5 must be able to close the logical channel and thus, free the channel for future use.

Enumerated Requirements Specifications.

SPEC_REQ_NUM_5.1 - Layer 5 must be able to initialize all requirements necessary to operate at this level.

SPEC_REQ_NUM_5.1.1 - Must be able to load from a storage device any information required by Layer 5 initialization which is subject to change. This is to facilitate changes, by requiring changes only to the information in the storage device and not to the program's source code.

SPEC_REQ_NUM_5.2 - Must be able to transmit a variable length NETOS message.

SPEC_REQ_NUM_5.2.1 - Should be able to map the logical resource name to the physical host process codes for both the source and destination.

SPEC_REQ_NUM_5.2.1.1 - If an invalid logical resource name, should return an error code.

SPEC_REQ_NUM_5.2.2 - Should establish a connection; if no errors, send the NETOS message over the channel; and then close the connection.

SPEC_REQ_NUM_5.3 - Must be able to receive a variable length NETOS message.

SPEC_REQ_NUM_5.3.1 - Should be able to map the logical resource name to the physical host process codes for both the source and destination.

SPEC_REQ_NUM_5.3.1.1 - If an invalid logical resource name, should return an error code.

SPEC_REQ_NUM_5.3.2 - Should establish a connection; if no errors, receive the NETOS message over the channel; and then close the connection.

SPEC_REQ_NUM_5.4 - Must be able to transmit a variable length file.

SPEC_REQ_NUM_5.4.1 - Should be able to map the logical resource name to the physical host process codes for both the source and destination.

SPEC_REQ_NUM_5.4.1.1 - If an invalid logical resource name, should return an error code.

SPEC_REQ_NUM_5.4.2 - Should establish a connection; if no errors, send the file over the channel; and then close the connection.

SPEC_REQ_NUM_5.4.2.1 - Should open the file.

SPEC_REQ_NUM_5.4.2.2 - Should transmit the file a block, of size MSIZE, at a time.

SPEC_REQ_NUM_5.4.2.3 - If the last data in the file is less than MSIZE, then adjust the block so that it is the same size as the data.

SPEC_REQ_NUM_5.4.2.4 - When the EOF is reached some how indicate to the receiving end that there is no more data.

SPEC_REQ_NUM_5.4.2.5 - Should close the file when done.

SPEC_REQ_NUM_5.5 - Must be able to receive a variable length file.

SPEC_REQ_NUM_5.5.1 - Should be able to map the logical resource name to the physical host process codes for both the source and destination.

SPEC_REQ_NUM_5.5.1.1 - If an invalid logical resource name, should return an error code.

SPEC_REQ_NUM_5.5.2 - Should establish a connection; if no errors, receive the file over the channel; and then close the connection.

SPEC_REQ_NUM_5.5.2.1 - Should create a file and leave open.

SPEC_REQ_NUM_5.5.2.2 - Should receive the file blocks at a time.

SPEC_REQ_NUM_5.5.2.3 - If the EOF has been reached, then close the file; otherwise continue receiving and writing blocks of data to the file.

Layer 6

Narrative Functional Specifications.

Layer 6 must be able to support the interface requirements defined in the Layer 7 protocol. This requires the capability to transmit and receive both request and status messages by calling on Layer 5. Layer 6 must also be able to support the transmission and reception of files.

And finally, Layer 6 should have the capability to send and receive strings of characters.

Enumerated Requirements Specifications.

SPEC_REQ_NUM_6.1 - Layer 6 must be able to initialize all requirements necessary to operate at this level.

SPEC_REQ_NUM_6.1.1 - Must be able to load from a storage device any information required by Layer 6 initialization which is subject to change. This is to facilitate changes, by requiring changes only to the information in the storage device and not to the program's source code.

SPEC_REQ_NUM_6.2 - Must be able to transmit a NETOS request message.

SPEC_REQ_NUM_6.2.1 - Must be able to build a NETOS request message from the input parameters.

SPEC_REQ_NUM_6.2.2 - If any input parameters are invalid, then return an error code.

SPEC_REQ_NUM_6.2.3 - Should return a status code indicating the status of the transmission.

SPEC_REQ_NUM_6.3 - Must be able to transmit a NETOS status message.

SPEC_REQ_NUM_6.3.1 - Must be able to build a NETOS status message from the input parameters.

SPEC_REQ_NUM_6.3.2 - If any input parameters are invalid, then return an error code.

SPEC_REQ_NUM_6.3.3 - Should return a status code indicating the status of the transmission.

SPEC_REQ_NUM_6.4 - Must be able to transmit a NETOS string.

SPEC_REQ_NUM_6.4.1 - Must be able to build a NETOS string message from the input parameters.

SPEC_REQ_NUM_6.4.2 - If any input parameters are invalid, then return an error code.

SPEC_REQ_NUM_6.4.3 - Should return a status code indicating the status of the transmission.

SPEC_REQ_NUM_6.5 - Must be able to transmit a NETOS file.

SPEC_REQ_NUM_6.5.1 - If any input parameters are invalid, then return an error code.

SPEC_REQ_NUM_6.5.2 - Should return a status code indicating the status of the transmission.

SPEC_REQ_NUM_6.6 - Must be able to receive a NETOS request message.

SPEC_REQ_NUM_6.6.1 - If any input parameters are invalid, then return an error code.

SPEC_REQ_NUM_6.6.2 - Must be able to extract the output parameters from the received NETOS request message.

SPEC_REQ_NUM_6.6.3 - Should return a status code indicating the status of the reception.

SPEC_REQ_NUM_6.7 - Must be able to receive a NETOS status message.

SPEC_REQ_NUM_6.7.1 - If any input parameters are invalid, then return an error code.

SPEC_REQ_NUM_6.7.2 - Must be able to extract the output parameters from the received NETOS status message.

SPEC_REQ_NUM_6.7.3 - Should return a status code indicating the status of the reception.

SPEC_REQ_NUM_6.8 - Must be able to receive a NETOS string.

SPEC_REQ_NUM_6.8.1 - If any input parameters are invalid, then return an error code.

SPEC_REQ_NUM_6.8.2 - Must be able to extract the output parameters from the received NETOS string message.

SPEC_REQ_NUM_6.8.3 - Should return a status code indicating the status of the reception.

SPEC_REQ_NUM_6.9 - Must be able to receive a NETOS file.

SPEC_REQ_NUM_6.9.1 - If any input parameters are invalid, then return an error code.

SPEC_REQ_NUM_6.9.2 - Should return a status code indicating the status of the reception.

III. Design, Implementation, and Testing

Introduction

Following software engineering principles, the system requirements are transformed into a detail design representation of the software and then the detail design is transformed to the implementation of the software code. Since there were relatively few changes to the requirements of the old NETOS system, the design is very similar to the design of the old system. The one major change is contributed to allowing variable length messages. The design and implementation of the new NETOS software shall be discussed beginning with the library package which supports the ISO Layers, followed by each ISO Layer, and then lastly the Central System. Appendix A contains the test plan for the software code and Appendix B contains the User's Manual. The structure charts, which were produced from the SADT diagrams, and the source code are kept on file by the LSINET manager, presently Dr. Hartrum.

NETOS Library

From the beginning, it became apparent that a separate library package containing various subroutines which perform simple and common operations would be helpful and would simplify the coding of the ISO Layers.

In order to allow the transmission of binary files and other data not confined to the standard ASCII set which utilizes 7 bits, a type that includes the extended ASCII set, which utilizes a full 8 bits, is required. Janus Ada provides such a type called BYTE. However a frame, packet, and message are made up of several bytes, thus an array of bytes was needed. Moreover, since the lengths of the frames, packets, and messages will be variable, a new type called BSTRING was designed which consisted of a record with two fields. One field contains an array of bytes, and the other field contains the current length of valid data in the array. Since arrays are required to be constrained by Ada, a maximum length had to be established for the array. The array had to be large enough to hold a maximum frame length. Two bytes of the frame were chosen to contain the frame length. Thus, the maximum frame length is 64K bytes. However, a frame of this size will resort in other hosts being deprived of the Central System due to the long transmission time required to transmit 64K bytes. So at Layer 6, a limit is arbitrarily set to bound the frame length to 510 bytes. Therefore, the maximum array length was arbitrarily set to 1024. This more than meets the limited maximum frame length while establishing an upper limit to the size of the string, since the string is converted to a BSTRING, that can be sent at the Layer 6 level.

Getting the time stamp proved to be more complicated than was expected. The CALENDAR package provided by Janus Ada was used. However, the time had to calculated into its various parts; hours, minutes, seconds, and fraction of seconds. These parts are transformed into strings. The strings are concatenated to form one string with the appropriate delimiter inserted between the parts. The final result is a string of length 11 containing the proper time stamp format: HH:MM:SS.FF

Several routines were designed to allow ease in the transforming of one type to another such as CHARACTER to BYTE. Also several routines were design to operate on the new type BSTRING. This includes concatenating items to form a BSTRING, deleting BYTES from a BSTRING, transforming BSTRINGS to STRINGS and STRING to BSTRING, and several variations of these with various types.

Layer 1

Layer 1 was functionally divided into five modules: initialize the port, write a byte to port, read a byte from the port, determine if port is ready to receive a byte for output and, determine if port has a byte ready for input. Since Layer 1 is responsible for the interfacing with the serial communication ports; low level routines are required. JANUS Ada provides two ways to access the communication ports. The first is through the use of a JANUS Ada

supported assembly language. The second is through the use of the UTIL library package supplied by Janus Ada. The second approach was taken in order to maintain the code at as high a level as possible and thus keep the code as simple as possible.

The UTIL library contains the procedure SIMPLE_INT_CALL which supports IBM ROM-BIOS service interrupts. Services 0-3 of interrupt 20 (14 hex) were used to initialize the ports, send a byte, receive a byte, and get the port status. Care had to be taken while using SIMPLE_INT_CALL as the port number had to be corrected since the ROM-BIOS identifies port 1 as port 0, port 2 as port 1, and so on. Also care had to be taken, since the register input to SIMPLE_INT_CALL was defined as integers, but the transmission of data had to be in bytes. The type BYTE allows full use of the entire 8-bit ASCII set, which is required in order to transmit binary files. The type CHARACTER can not support a full byte of data as it is restricted to a 7-bit ASCII set, while the type INTEGER is less restricted and thus would result in errors if an attempt was made to transmit a value greater than 256 (8-bits).

The initialize routine and the two status routines provide the calling module with status of its operation. Following the old NETOS software, the initialize routine supports only baud rate of 1200, 2400, 4800, 9600 bits per

second. Baud rates other than these or an error while initializing the port will be reported back to the calling module. The two status routines notify the calling module whether the port is ready or not, thus they return a boolean.

Layer 2

Due to the size of the Layer 2 software, two subpackages support the Layer 2 package; one for the send portion and one for the receive portion. These three packages will be described in the following order: first, the send portion; next, the receive portion; and finally, the overall Layer 2 package.

Send Portion.

The send portion basically provides for the sending of a frame. This is divided into five modules: calculate a checksum, build the frame, handle the NETOS handshake, send the frame, and receive the acknowledgement.

The checksum is calculated in the same manner as in the old NETOS software. All the bytes of the packet are added up and the lower byte of the result is returned as the checksum. Since the LSINET has proven to be very reliable, this simple checksum provide ample error detection capability.

The frame is built by simply concatenating the various parts to form one BSTRING. An ASCII SOH indicates the

beginning of the frame. In order to support variable length frames, a frame length field of two bytes is provided. This allows a frame to be the maximum length of 64K bytes. The variable length packet and its checksum are included in the frame. And finally, an ASCII ETX indicates the end of the frame.

After examining this format on the protocol analyzer it became apparent that delimiters were required to separate the different fields. It was very hard to distinguish which bytes belonged to which field without having to count the individual bytes in order to determine its position. Delimiters separate the various fields and thus provide easier recognition of the data. Some throughput is lost but is regained since this format uses only one ASCII ETX while the old format utilized five ASCII ETXs. In contrast with the old system which used five consecutive ASCII ETXs to indicate the end of data in the event the original frame length was altered, a timeout interval is used in receiving each byte. If the timeout interval expires, then the checksum verification should fail, a NAK returned to the sender, and a reattempt made in sending the frame.

The LSINET handshake requires that the source sends a transmit request (TR) and the receiving end sends back a transmit acknowledge (TA) before data can be transmitted. This software uses the same TR , an ASCII B, and the same

TA, an ASCII A, as the old software. A timeout interval is provided to ensure a hang up does not occur. The timeout interval is approximately 3.7 seconds. The results are returned to the calling module. Possible result conditions are; the handshake was successful, the timeout occurred before a TR was able to be sent, a TA was not received within the timeout interval, or something other than a TA was received.

The frame is sent out the port byte-by-byte. Again a timeout interval is provide to ensure the system does not hang up. If the timeout expires, the sending of the frame is aborted. The receiving end should return a NAK which will signal the Layer 2 package to try sending the frame another time.

Receive Portion.

The receive portion provides for the receiving of a frame. This is divided into five modules: handle the NETOS handshake, receive the frame, verify the checksum, send back an acknowledgement, and extract the packet from the received frame. receive the acknowledgement.

The results of the handshake attempt are returned tc the calling module. Possible result conditions are; the handshake was successful, the timeout occurred before a TR was received, something other than a TR was received, or a TR was received but the timeout occurred before a TA could

be sent back.

The frame is received by first examining the frame length field and attempting to receive that many bytes. A timeout is provided to ensure no hang up occurs. The frame is built by concatenating the bytes to reform the original sent frame.

Possible results that can be returned to the calling module are as follows; the first byte was not a ASCII SOH, the timeout occurred before the entire frame was received, or the entire frame was received.

The verify checksum module calculates the checksum of the received packet and compares it with the checksum that was sent in the frame. A boolean is returned; TRUE if they match, FALSE if they do not match.

Based on the results of the verify checksum module, an ASCII ACK is sent back if the checksums matched, or an ASCII NAK is sent back if the checksums did not match.

Last, the packet is extracted from the frame by deleting the frame header bytes, the SOH and frame length field, and deleting the frame trailer bytes, the checksum and the ETX.

Overall Layer 2 Package.

The Layer 2 package provides three functions: initialize Layer 2, send a packet, and receive a packet. Although Layer 2 does not require any initialization, the

initialize Layer 2 module, which simply calls the initialize Layer 1 module, is required in order to maintain the ISO 7-Layer model. The send module follows the old NETOS software of allowing three attempts to establish a LSINET handshake and three attempts to try to send a frame. Multiple attempts are made due to the assumption that the transmission across the network is unreliable. Three attempts was arbitrarily chosen. It does this by calling on the modules in the send subpackage described above. The receive module receives a frame and extracts the packet by calling on the modules in the receive subpackage described above.

Layer 3

This layer is divided into three main functions: initialize Layer 3, send a message, and receive a message. A global table is provided to all three modules. This table maps the SRC/DST codes to the port number which is linked to that SRC/DST host and the baud rate for that port. This table is configured as an array of records, where the record contains the three fields; SRC/DST code, port number, and baud rate. Table III list the SRC/DST codes.

TABLE III.
SRC/DST Codes

A	-	Work Station A
B	-	Central System
C	-	Work Station C
D	-	Work Station D
E	-	Work Station E
F	-	Work Station F
G	-	Work Station G
H	-	Work Station H
I	-	Work Station I
J	-	Work Station J
*	-	Any Host (don't care)

The initialize Layer 3 module calls on a submodule to build the global table from data contained in a file PTABLE.DAT. This allows any changes to the port routing and configuration to be as simple as making a change to file. Figure 5 contains a listing of PTABLE.DAT. Once the table is built, each entry is passed to the initialize Layer 2 module in order to initialize the ports.

```
A    1    9600
B    1    9600
C    1    9600
D    1    9600
E    1    9600
F    1    9600
G    1    9600
H    1    9600
I    1    9600
J    1    9600
```

Figure 5. PTABLE.DAT

The send message module builds a packet by concatenating the various fields to form a packet. The packet consists of the SRC/SPN/DST/DPN codes, the system message byte, the message, and the time stamp. The system byte indicates whether this message is a Layer 3 system message, presently there are no Layer 3 system message defined. An ASCII 1 indicates a Layer 3 system message while an ASCII 0 indicates a normal message. Presently a Layer 3 system can only be sent by setting the appropriate parameter in the Layer 6 SEND_NETOS_STRING module. After the packet has been built, the DST code is sent to a submodule which looks up in the global table the correct port to route the packet on. The packet is then sent by calling on Layer 2.

The receive message module is divided into two parts; receive a message from a particular host, and receive a message from any host. The SRC byte determines which course to take, where an ASCII * indicates receive from any host.

When receiving a message from a particular host, the SRC code is passed to a submodule to locate the correct port to receive, attempts are made to receive a packet until a packet is received without error (since this is a non-interrupt version, Layer 3 continues 'listening' until a valid packet is received), the SRC and DST codes of the packet are evaluated to see if they match with what was expected. If they do not match, then discard the packet and

try again. If they do match, then extract the message from the packet by deleting the packet header and trailer, evaluate the system message byte to determine whether to pass the message to the process system message module or to pass the message back to the calling module.

When receiving a message from any host, the entries in the global table are polled, the host codes are looked up along with port number associated with it, and an attempt is made to receive a packet on that port. If an error occurs or no packet was found, then continue to poll by trying the next entry in the table. If a packet was received then extract the message from the packet by deleting the packet header and trailer, evaluate the system message byte to determine whether to pass the message to the process system message module or to pass the message back to the calling module.

Layer 4

This layer is divided into six main functions: initialize Layer 4, establish a connection from a sender point of view, establish a connection from a receiver point of view, close a connection, send messages, and receive messages. A global table is provided to all the modules. This table contains the channel assignments. It maps the SRC/SPN/DST/DPN codes to a particular channel number. There are presently a total of nine channels. The limit of nine

channels was chosen since there are nine workstations presently on the network. This table is configured as an array of records, where the record contains the six fields; SRC, SPN, DST, DPN, channel number, and a used field.

The initialize Layer 4 module build the global table, initiating it so all channels are available, and then calls the initialize Layer 3 module.

The module which establishes a connection from a sender point of view first calls a submodule to to locate a free channel. This submodule first counts all the used channels. A maximum of nine channel was arbitrarily chosen since there are nine workstations on the network. If the count equal to nine, it then returns a error indicating all channels are in use. Otherwise a channel is assigned.

If the submodule returns no errors, a request channel message, a 1-byte message containing a ASCII R, is sent to the desired destination. If the destination is able to set up a connection at its end it will return a valid channel response message, a 1-byte message containing a ASCII V, otherwise it will return an invalid channel response message, a 1-byte message containing an ASCII I. Once the response message is decoded, the channel is either returned to the calling module or its is freed and an error is returned indicating the problem.

The module for establishing a connection from a

receiver stand point first receives a message, and then determines if it is a request channel message. If the message is not a request channel message, then try getting anther message. If message is a request channel message, then it is evaluated to see if the message is for the correct connection. If for the correct connection, then call the submodule to assign a channel. If a channel can be assigned then, return a valid channel reply to the source and return the channel number to the calling module. If a channel can not be assigned, then return an invalid channel reply to the source and return the error code indicating the problem to the calling module.

The close connection module looks up the channel in the global table and deletes the assignment thus freeing the channel. If the channel could not be found then an error code is returned otherwise a no error code is returned. There is no handshaking requirement with the other end. Each end must independently terminate its own connection. This is accomplished by Layer 5.

The send messages module first looks up the SRC/SPN/DST/DPN codes in the global table from the channel number. These codes are used by Layer 3 to build the packet. The message is then broken off msize, msize is the variable length message size specified by the application program, bytes at a time and sent submessage by submessage until the original message is less than or equal to msize

bytes. At this time the remainder of the message is sent and then a message of 25 null bytes is sent indicating the end of message to the destination. The special end of message consisting of 25 null bytes was arbitrarily chosen.

The receive message module first receives a message, and then compares the message SRC/SPN/DST/DPN codes with that of the channel number. If they do not match up, then try getting another message. If they do match up, then continue building back the original message from these submessages until an end of message is received.

Layer 5

The Layer 5 package contains five main modules: initialize Layer 5, send a NETOS message, receive a NETOS message, send a file, and receive a file. A global table is provided which maps the logical host names with the physical host/process codes. This table is configured as an array of records, where each record contains three fields: one for the logical host name, a string of length 10; another for the host code, a character; and the last for the process number, an integer from 1-9.

The initialize Layer 5 module builds the global table from data in the file HTABLE.DAT. This simplifies all changes to the host assignments as simply a matter of changing data in a file.

The send NETOS message module simply looks up the

source and destination codes from the global table, establishes a connection, sends the NETOS message, and then closes the connection. The status is returned to the calling module indicating if there were any problems or if the NETOS message was sent successfully.

The receive NETOS message module simply looks up the source and destination codes from the global table, and establishes a connection. If attempting to receive a NETOS message from any source, the Layer 4 establish connection module will return the SRC/SPN/DST/DPN codes for this connection. These codes are then mapped back to the physical host names from the global table. The NETOS message is then received, and the connection is then closed. The status is returned to the calling module indicating if there were any problems or if the NETOS message was received successfully.

The send file module is basically the same as the send NETOS message module except after a connection is established, the file is opened and data from the file is read and sent a block, of length msize, at a time. When the there is less than a block of data left, the remaining data is sent and then a block of 50 null bytes is sent to indicate the end of file to the destination. After this, the file is closed and connection is closed. If there were any errors, the file and connection both are closed and the

error code is returned to the calling module.

The receive file module is also basically the same as the receive NETOS message module except after the connection is established, a file is created and data is received and read to the file until a end of file block is received. After this, the file is closed and the connection is closed. If there were any errors, the file and connection both are closed and the error code is returned to the calling module.

Layer 6

There are seven main modules at the Layer 6 level: initialize layer 6; and send and receive, NETOS request, NETOS status, NETOS string, and NETOS file. All input parameters are checked for validity. The valid parameters are listed in Appendix B, the user's manual. Although, the initialize Layer 6, send NETOS file, and receive NETOS file modules do nothing more that call their corresponding Layer 5 modules, they are required in order to maintain the ISO 7-Layer model.

The send modules simply build a NETOS message by concatenating the input parameters to form the NETOS message and then send the NETOS message by calling on Layer 5.

Likewise the receive modules simply extract the output parameters from the received NETOS message and pass the parameters to the calling module.

Central System

Since the Central System is an LSI-11/23 microcomputer and as such does not support Ada, the coding had to remain in the programming language C. Therefore the old NETOS source code was modified rather than rewritten. Unfortunately, the only available listing of the source code was for Version 1.8 while the currently running version was 1.9. Therefore, more modifications had to be made than was expected just to bring the system up to the present version.

There were two major changes required to bring the system up to date. The port vectors had to be updated. This was accomplished by examining the file PTABLE.DAT for the new vectors. Also, the hosts designated as monitor stations were reassigned to the hosts specified in the latest version.

Once the software was brought up to date, the modifications to allow variable length frames were accomplished. The main routine was modified by eliminating the prompt to the user for varying the message size in the initialize central system module. This included eliminating the parameter 'msize' passed when calling the initialize central system module. This was no longer needed since the frame will contain the frame length. Also, the version number displayed on the monitor was changed to Version 2.0.

Rather than allocating space for a packet, receiving a frame, converting it to a packet, then converting the packet

back to a frame, and transmitting the frame; it appeared much simpler to leave everything at the frame level. Since the frames are variable in length, the maximum space required by a frame is allocated, the frame is received, and then retransmitted to the destination. No changes were required at the Layer 1 level.

The ISO Layer 2 receive module had to be changed to handle variable length frames. The frame length field of the received frame must be examined in order to determine how many bytes to receive. Therefore, the receive frame module was changed to first examine the frame length and then receive the specified number of bytes. The verify checksum module was modified to allow variable length frames. If a frame is successfully received, then the entire frame is stored in a buffer and a pointer to the buffer is passed to the ISO Layer 2 send module.

The ISO Layer 2 send module also had to be changed to handle variable length frames. First the frame size is calculated by examining the frame length field, and then the specified number of bytes from the buffer are transmitted.

A modification was also made to correct the problem of error messages from previous transmitted frames remaining on the monitor of the Central System. This was corrected by allowing a message to be displayed for frames received with no errors. Thus the 'no error' message overwrites the

previous error message. Therefore, the message displayed on the monitor always corresponds to the last received message.

IV. Conclusions and Recommendations

Conclusions

The goal of this thesis effort was to correct the various discrepancies found in the previous software, make the necessary modifications to allow variable length messages, and write the code in the Ada programming language. The conclusions to these three tasks are described as follows.

The task which required the least amount of effort was correcting the discrepancies found in the previous software. The problem areas were neither large in scope or complex. It appears most discrepancies were probably due to a lack of communication with other groups working on related modules or due simply to carelessness. More time was taken in discovering the discrepancies than was taken to actually correct them.

The task of making modifications to allow variable length messages proved to be more difficult than was expected. Modifications had to be made both on the software for the worstations, implemented in Ada, and on the software for the central system, implemented in C. Extra effort was required not only because of the two different languages but also because there was no documentation or source code for the current operating version, Version 1.9, of the software

running on the central system.

The task of implementing the software in Ada proved to be the most time consuming. The length of the code grew to be quite large. However by maintaining the ISO 7-Layer Model and 'packaging' the code into the appropriate layers, the task did not become overwhelming. A library package was developed, NETLIB, to provide modules which performed common routines for all the layers. Fortunately, Janus/Ada had a library package, DOSCALL, which provided low-level I/O routines. A type BSTRING was defined to allow a variable length array of type BYTE. This allowed for easy manipulation of bytes to frames to packets to messages. The hard typing of Ada had to be overcome as there were frequent requirements to convert from one type to another. However, this was remedied by providing these common conversion routines in the library package NETLIB.

Recommendations

There are an endless number of future enhancements and research projects which could follow up on this thesis effort. Some areas include user application software, performance evaluation, and flow control.

Presently there are no application programs utilizing this network. Developing application programs such as a Printer Spooler, a Mass Storage System, or a Database Management System would not only improve the support of the

software systems laboratory but would also provide a realistic project in applying software engineering techniques.

A performance analysis of the network should be conducted to measure performance parameters and analyze flow of traffic and delay throughout the network. From this analysis, improvements can be made to increase network performance.

A potential problem exists with the way the end-of-message and end-of-file are handled. For instance, if a file contains 50 consecutive nulls, a string of 50 consecutive nulls is used as the end-of-file indicator, and the variable message length is specified as 50, then there exists the chance that the 50 consecutive nulls in the file will be blocked off as a message length of 50. This will mislead the receiver into beleiving that this data is the end-of-file indicator. Thus, the receiver will not attempt to receive the rest of the file. This problem is currently be worked on by EENG 7.93, Advanced Software Engineering.

Now that the network allows for variable length message, the next step to take would be to let the network control the flow of traffic through the network by monitoring the traffic and dictating to the worstations the maximum frame size they can transmit. When traffic is low, the maximum frame size can be increased. When traffic is

high, the maximum frame size should be decreased to allow fair access to the network by all users. The dictating of the maximum size could be handled with a Layer 4 System Message, similar to the Layer 3 System Message, since Layer 4 is responsible for dividing the data to be transmitted up into messages.

APPENDIX A

Test Plan

Testing consisted of three phases; testing of individual modules, testing of each layer, and system integration testing. As each module was being coded, it was tested. Once all the modules which comprise a layer successfully passed their tests, the layer as a whole underwent testing. Each layer was built and tested on lower layers which had successfully passed their tests. Once all the layers had been tested, the entire system was tested. The test plan and the test results are listed by layers starting with Layer 1 and finishing with Layer 6.

TEST PLAN 29 OCT 88

for Layer 1

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
1. Test Initialize_ Layer_1		
a. Enter a negative integer for port.	a. Should return status code #102.	a. Passed
b. Enter the integer: 0 for the port.	b. Should return status code #102.	b. Passed
c. Enter the integer: 1 for the port.	c. Should return status code #0.	c. Passed
d. Enter the integer: 2 for the port.	d. Should return status code #0.	d. Passed
e. Enter a integer value greater than 2 for the port.	e. Should return status code #102.	e. Passed
f. Enter a negative integer for the baud	f. Should return status code #101.	f. Passed
g. Enter the integer: 0 for the baud.	g. Should return status code #101.	g. Passed
h. Enter the integer: 1200 for the baud.	h. Should return status code #0.	h. Passed
i. Enter the integer: 2400 for the baud.	i. Should return status code #0.	i. Passed
j. Enter the integer: 4800 for the baud.	j. Should return status code #0.	j. Passed
k. Enter the integer: 9600 for the baud.	k. Sturn status code #0.	k. Passed
l. Enter an integer other than 1200,2400 4800, 9600 for the baud.	l. Should return status code #101.	l. Passed
2. Test Input_Port. Sending various data to the ports.		
a. Enter the integer: 1 for the port.	a. Should return the byte read from port 1	a. Passed
b. Enter the integer: 2 for the port.	b. Should return the byte read from port 2	b. Passed

TEST PLAN 29 OCT 88

for Layer 1

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
3. Test Output_Port. Output various data.		
a. Enter the integer: 1 for the port.	a. Should write the byte to port 1.	a. Passed
b. Enter the integer: 2 for the port.	b. Should write the byte to port 2.	b. Passed
4. Test Input_Port_Not_Ready. NOTE: For a-e the port is set clear to receive.		
a. Enter a negative integer for the port	a. Should return boolean of FALSE.	a. Passed
b. Enter the integer: 0 for the port.	b. Should return boolean of FALSE.	b. Passed
c. Enter the integer: 1 for the port.	c. Should return boolean of TRUE.	c. Passed
d. Enter the integer: 2 for the port.	d. Should return boolean of TRUE.	d. Passed
e. Enter an integer value greater than 2 for the port. NOTE: For f-g the port is not set clear to receive.	e. Should return boolean of FALSE.	e. Passed
f. Enter the integer: 1 for the port.	f. Should return boolean of FALSE.	f. Passed
g. Enter the integer: 2 for the port.	g. Should return boolean of FALSE.	g. Passed

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
5. Test Output_Port_Not_Ready. NOTE: For a-e the port is set clear to receive.		
a. Enter a negative integer for the port	a. Should return boolean of FALSE.	a. Passed
b. Enter the integer: 0 for the port.	b. Should return boolean of FALSE.	b. Passed
c. Enter the integer: 1 for the port.	c. Should return boolean of TRUE.	c. Passed
d. Enter the integer: 2 for the port.	d. Should return boolean of TRUE.	d. Passed
e. Enter an integer value greater than 2 for the port. NOTE: For f-g the port is not set clear to receive.	e. Should return boolean of FALSE.	e. Passed
f. Enter the integer: 1 for the port.	f. Should return boolean of FALSE.	f. Passed
g. Enter the integer: 2 for the port.	g. Should return boolean of FALSE.	g. Passed

TEST PLAN 29 OCT 88

for Send Portion of Layer 2

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
1. Test for Calc_Checksum.		
a. Enter a null packet.	a. Should return a byte of value 0.	a. Passed
b. Enter variable length packets with variable data.	b. Should return a byte of value equal to the lower byte result of the sum of all the data bytes.	b. Passed
2. Test Build_Frame.		
a. Enter the various values data for the packet, psize, and checksum.	a. Should return a bstring of the proper frame format.	a. Passed
3. Test Send_TR_Wait_TA		
a. Set port not ready.	a. Should return status code #201.	a. Passed
b. Set port ready, but do not transmit a reply to the port.	b. Should output a 'B' through the port, and should return status code #202.	b. Passed
c. Set port ready, and transmit a 'A' reply	c. Should output a 'B' through the port, and should return status code #0.	c. Passed
d. Set port ready, and transmit any reply other than 'A'.	d. Should output a 'B' through the port, and should return status code #203.	d. Passed

for Send Portion of Layer 2

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
4. Test Send_Frame.		
a. Enter the frame.	a. Should output the frame byte-by-byte through the port.	a. Passed
5. Test Wait_for_ACK_NAK		
a. Do not transmit a reply to the port.	a. Should return status code #204.	a. Passed
b. Transmit an ASCII ACK to the port.	b. Should return status code #0.	b. Passed
c. Transmit an ASCII NAK to the port.	c. Should return status code #205.	c. Passed
d. Transmit something other than an ASCII.ACK or NAK	d. Should return status code #206	d. Passed

TEST PLAN 29 OCT 88

for Receive Portion of Layer 2

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
1. Test Wait_for_TR_ Send_TA.		
a. Do not transmit a 'B' to the port.	a. Should return status code #211.	a. Passed
b. Transmit anything other than a 'B' to the port.	b. Should return status code #212.	b. Passed
c. Set output port not ready and transmit a 'B' to the port.	c. Should return status code #213.	c. Passed
d. Set output port ready and transmit a 'B' to the port.	d. Should output a 'A' through the port, and return a status code #0.	d. Passed
2. Test Receive_Frame.		
a. Send less than 4 bytes of a frame.	a. Should return a null frame and a status code #214.	a. Passed
b. Transmit a frame that does not begin begin an ASCII.SOH.	b. Should return a null frame and a status code #215.	b. Passed
c. Transmit a frame that begins with an ASCII.SOH, send at least 4 bytes but not all the frame.	c. Should return the frame received at that point and a status code #216.	c. Passed
d. Transmit a frame that begins with an ASCII.SOH, send the entire frame.	d. Should return the frame and a status code #0.	d. Passed

TEST PLAN 29 OCT 88

for Receive Portion of Layer 2

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
3. Test Verify_Checksum.		
a. Enter a frame with correct checksum.	a. Returns a boolean TRUE.	a. Passed
b. Enter a frame with an incorrect checksum	b. Returns a boolean FALSE.	b. Passed
4. Test Send_ACK_NAK.		
a. Set port not ready.	a. Aborts sending ack/nak.	a. Passed
b. Port ready, byte value set equal to ASCII.ACK.	b. Should output an ASCII.ACK to the port	b. Passed
c. Port ready, byte value set equal to ASCII.NAK.	c. Should output an ASCII.NAK to the port	c. Passed
5. Test Extract_Packet_From_Frame.		
a. Input various frames	a. Removes the frame header and trailer, returns the packet.	a. Passed

TEST PLAN 29 OCT 88

for Layer 2

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
1. Test the sending of a packet.		
a. Do not allow proper NETOS handshake.	a. Should return status code of handshake error.	a. Passed
b. Allow proper NETOS handshake on the 1st try.	b. Should send the frame byte-by-byte. Status code #0.	b. Passed
c. Allow proper NETOS handshake after the 1st try.	c. Should send the frame byte-by-byte. Status code #0.	c. Passed
d. Allow proper NETOS handshake after the 2nd try.	d. Should send the frame byte-by-byte. Status code #0.	d. Passed
e. Allow proper NETOS handshake after the 3rd try.	e. Should return status code of handshake error.	e. Passed
f. Do not allow an ACK reply.	f. Should return status code of ACK/NAK error	f. Passed
g. Allow proper ACK reply on the 1st try	g. Should return status code #0.	g. Passed
h. Allow proper ACK reply after the 1st try.	h. Should return status code #0.	h. Passed
i. Allow proper ACK reply after the 2nd try.	i. Should return status code #0.	i. Passed
j. Allow proper ACK reply after the 3rd try.	j. Should return status code of ACK/NAK error	j. Passed

TEST PLAN 29 OCT 88

for Layer 2

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
2. Test receiving of a packet.		
a. Do not allow proper NETOS handshake.	a. Should return status code of handshake error, and a null packet.	a. Passed
b. Allow proper NETOS handshake. Receive a packet with error.	b. Should return status code of receive error and a null packet.	b. Passed
c. Allow proper NETOS handshake. Receive a packet with bad checksum.	c. Should reply with a a NAK, and return a null packet.	c. Passed
d. Allow proper NETOS handshake. Receive a packet with good checksum.	d. Should reply with a an ACK and return the packet.	d. Passed

TEST PLAN 29 OCT 88

for Layer 3

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
1. Test Build_Port_ Array.		
a. Let PTABLE.DAT file vary in data and length.	a. Array should contain the same data that was in the file.	a. Passed
2. Test Determine_Port_ address.		
a. Let SRC/DST code be the 1st entry in the table.	a. The corresponding port id should be returned along with a status code #0.	a. Passed
b. Let SRC/DST code be the last entry in the table.	b. The corresponding port id should be returned along with a status code #0.	b. Passed
c. Let SRC/DST code be somewhere in the middle of the table.	c. The corresponding port id should be returned along with a status code #0.	c. Passed
d. Let SRC/DST code not be in table.	d. A port id of #0 is returned along with a status code #301.	d. Passed
3. Test Initialize_ Layer_3.		
a. Vary the array length and use valid port and baud data.	a. A status code of #0 should be returned.	a. Passed
b. Let one of the port data be invalid.	b. A port error should be returned as the status code.	b. Passed
c. Let one of the baud data be invalid.	c. A baud error should be returned as the status code.	c. Passed

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
4. Test Send_Message.		
a. Let DST code not be in the table.	a. A status code of #301 should be returned.	a. Passed
b. Let DST code be in the table and vary SRC/DST/DST/DPN, message, msize, and the system byte.	b. A status code of #0 should be returned and the packet should be built and sent.	b. Passed
5. Test Validate_SRC_ DST.		
a. Let SRC code of the packet not be what is expected.	a. A status code of #302 should be returned.	a. Passed
b. Let the expected SRC code be the wildcard ASCII.*.	b. A status code of #0 should be returned.	b. Passed
c. Let SRC code of the packet be what is expected.	c. A status code of #0 should be returned.	c. Passed
d. Let DST code of the packet not be what is expected.	d. A status code of #303 should be returned.	d. Passed
e. Let the expected DST code be the wildcard ASCII.*.	e. A status code of #0 should be returned.	e. Passed
f. Let DST code of the packet be what is expected.	f. A status code of #0 should be returned.	f. Passed
6. Test Extract_Message _From_Packet.		
a. Vary the packet.	a. Should return the message, SRC/DST/SPN/ DST codes, and the system_msg byte.	a. Passed

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
7. Test Receive_Message		
a. Let the SRC code be the wildcard ASCII.* and not a system_msg (system_msg byte=0)	a. Should return any message received with the SRC/SPN/DST/DPN codes and status code of #0.	a. Passed
b. Let the SRC code be the wildcard ASCII.* and a system_msg (system_msg byte=1)	b. Should call the layer 3 system_msg routine.	b. Passed
c. Let the SRC code not be the wildcard ASCII.* and not a system_msg.	c. Should return the expected message with the SRC/SPN/DST/DPN codes and status code of #0.	c. Passed
d. Let the SRC code not be the wildcard ASCII.* and a system_msg.	d. Should call the layer 3 system_msg routine.	d. Passed

TEST PLAN 29 OCT 88

for Layer 4

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
1. Test Build_Channel_ Table.		
a. Build the table.	a. Table built with initial values.	a. Passed
2. Test Establish_ Channel_Num.		
a. Let all channels be in use.	a. Returns a channel # 0 and status code 402.	a. Passed
b. Let a channel be available, but one end of the connect-ion already in use.	b. Returns a channel # 0 and status code 401.	b. Passed
c. Let a channel be available and both ends of the connect-ion not in use.	c. Returns the next available channel # and a status code 0.	c. Passed
3. Test Connect.		
a. Let there be an establish channel error.	a. Returns a channel # 0 and an establish error.	a. Passed
b. Let there be a send message error.	b. Returns a channel # 0 and a send msg error.	b. Passed
c. Let the reply be invalid.	c. Returns a channel # 0 and a status code 403	c. Passed
d. Let the reply be valid.	d. Returns the next available channel # and a status code 0.	d. Passed

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
4. Test Listen.		
a. Receive an invalid channel request.	a. Returns a channel # 0 and a status code 404	a. Passed
b. Let there be a establish error.	b. Returns a channel # 0 and an establish error	b. Passed
c. Let there be a send message error.	c. Returns a channel # 0 and a send msg error.	c. Passed
d. Do not introduce any errors.	d. Returns the next available channel # and a status code 0.	d. Passed
5. Test Close.		
a. Try closing a channel not in the table.	a. Returns status code 405.	a. Passed
b. Try closing channel in the table.	b. Clears the entry in the table and returns a status code 0.	b. Passed
6. Test Send.		
a. Let there be an error in send msg.	a. Returns the error.	a. Passed
b. Send various length messages and msize.	b. Sends the message in lengths of msize followed by an EOM. Returns a status code 0.	b. Passed
7. Test Receive.		
a. Receive various lengths messages	a. Received messages followed by an EOM. Returns a status code 0.	a. Passed

TEST PLAN 29 OCT 88

for Layer 5

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
1. Test Determine_Host_Codes.		
a. Enter an invalid Host name.	a. Returns a status code 501.	a. Passed
b. Enter a vaild Host name.	b. Returns the host and process codes and a status code 0.	b. Passed
2. Test Determine_Host_Name.		
a. Enter invalid host and/or process codes	a. Returns a status code 502.	a. Passed
b. Enter a valid host and process code.	b. Returns the host name and a status code 0.	b. Passed
3. Test Read_A_Block_From_File		
a. Try a file size less than the block size.	a. Puts the data in a block equal to the data size and set the EOF flag.	a. Passed
b. Try a file size equal or greater than the block size.	b. Puts a block size amount of data in the block.	b. Passed
4. Test Write_A_Block_To_File.		
a. Let the block be a EOF block.	a. Sets the EOF flag.	a. Passed
b. Try various blocks.	b. Writes the block of data to the file.	b. Passed

TEST PLAN 29 OCT 88

for Layer 5

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
5. Test Initialize_ Layer_5		
a. Try different data in HTABLE.DAT	a. Loads the data in a Host Table	a. Passed
6. Test Send_Netos_Msg.		
a. Let there be a SRCE error.	a. Returns the error code.	a. Passed
b. Let there be a DEST error.	b. Returns the error code.	b. Passed
c. Let there be a Connect error.	c. Returns the error code.	c. Passed
d. Let there be a Send error.	d. Returns the error code.	d. Passed
e. Let there be a Close error.	e. Returns the error code.	e. Passed
f. Do not introduce any errors.	f. Returns a status code 0.	f. Passed
7. Test Receive_Netos_ Message.		
a. Let there be a SRCE error.	a. Returns the error code.	a. Passed
b. Let there be a DEST error.	b. Returns the error code.	b. Passed
c. Let there be a Listen error.	c. Returns the error code.	c. Passed
d. Let there be a SRC/ SPN error.	d. Returns the error code.	d. Passed
e. Let there be a DST/ DPN error.	e. Returns the error code.	e. Passed
f. Let there be a Receive error.	f. Returns the error code.	f. Passed
g. Let there be a Close error.	g. Returns the error code.	g. Passed
h. Do not introduce any errors.	h. Returns a status code 0.	h. Passed

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
8. Test Send_File.		
a. Input various errors	a. Returns the error code.	a. Passed
b. Try various files and Msize.	b. Sends the file by blocks of Msize followed by a EOF block. Returns a status code 0.	b. Passed
9. Test Receive_File.		
a. Input various errors	a. Returns the error code.	a. Passed
b. Receive various files.	b. Writes the received data to a local file. Returns a status code 0.	b. Passed

TEST PLAN 29 OCT 88

for Layer 6

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
1. Test Check_Msg_Type.		
a. Input an invalid Msg_Type.	a. Returns a status code 601.	a. Passed
b. Input a valid Msg_ Type.	b. Returns a status code 0.	b. Passed
2. Test Check_SRCE_DEST		
a. Input an invalid SRCE_DEST.	a. Returns a status code 602.	a. Passed
b. Input a valid SRCE_ DEST.	b. Returns a status code 0.	b. Passed
3. Test Check_Filename.		
a. Input an invalid Filename.	a. Returns a status code 603.	a. Passed
b. Input a valid Filename.	b. Returns a status code 0.	b. Passed
4. Test Check_Priority.		
a. Input an invalid Priority.	a. Returns a status code 604.	a. Passed
b. Input a valid Priority.	b. Returns a status code 0.	b. Passed
5. Test Check_Dest_ Device.		
a. Input an invalid Dest_Device.	a. Returns a status code 605.	a. Passed
b. Input a valid Dest_ Device.	b. Returns a status code 0.	b. Passed

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
6. Test Check_Status_ Character.		
a. Input an invalid Status_Character.	a. Returns a status code 606.	a. Passed
b. Input a valid Status _Character.	b. Returns a status code 0.	b. Passed
7. Test Send_NETOS_ Request.		
a. Input various errors	a. Returns the error.	a. Passed
b. Do not input errors.	b. Sends the message. Returns a status code 0.	b. Passed
8. Test Send_NETOS_ Status.		
a. Input various errors	a. Returns the error.	a. Passed
b. Do not input errors.	b. Sends the message. Returns a status code 0.	b. Passed
9. Test Send_NETOS_ File.		
a. Input various errors	a. Returns the error.	a. Passed
b. Do not input errors.	b. Sends the file. Returns a status code 0.	b. Passed
10. Test Send_NETOS_ String.		
a. Input various errors	a. Returns the error.	a. Passed
b. Do not input errors.	b. Sends the string. Returns a status code 0.	b. Passed

TEST CONDITIONS	EXPECTED ACTION	TEST RESULTS
11. Test Receive_Netos_ Request.		
a. Input various errors	a. Returns the error.	a. Passed
b. Do not input errors.	b. Receives the message. Returns a status code 0.	b. Passed
12. Test Receive_NETOS_ Status.		
a. Input various errors	a. Returns the error.	a. Passed
b. Do not input errors.	b. Receives the message Returns a status code 0.	b. Passed
13. Test Receive_NETOS_ File.		
a. Input various errors	a. Returns the error.	a. Passed
b. Do not input errors.	b. Receives the file. Returns a status code 0.	b. Passed
14. Test Receive_NETOS_ String.		
a. Input various errors	a. Returns the error.	a. Passed
b. Do not input errors.	b. Receive the string. Returns a status code 0.	b. Passed

APPENDIX B

USER GUIDE

Hardware Components

The hardware components for which the present NETOS system is configured for consist of the Central System (System B) and nine Z-248 microcomputer Work Stations (Systems A and C-J) on the LSINET. Also, a protocol analyzer (Hewlett-Packard 4955A) is provided, at present, in series between the Central System and System F for use in analyzing the network.

At present, the Work Stations are wired to communicate on Comm Port #2. Any reconfiguration of the wiring to another Comm Port will require changes to the PTABLE.DAT file. All links between the Central System and the Work Stations consist of serial RS-232 cables.

Software Components

The software written for the Work Stations was implemented in JANUS/Ada Version 2.0.2 on Z-248 microcomputers running a MSDOS Version 3.20 Operating System. And, the software written for the Central System was implemented in Whitesmith C on a LSI-11/23 microcomputer running a RT-11 Version 5.1 Operating System.

Getting Started

WARNING: MAKE SURE THE FLOPPY DISK IS NOT IN THE DISK DRIVE WHEN TURNING THE SYSTEM ON OR OFF. DAMAGE MAY OCCUR TO THE DATA ON THE FLOPPY DISK.

The network must be activated at the Central System before any communication can take place. The Central System is powered on and off by the switch on the power strip located on the floor along the wall next to the Central System (the LSI next to the wall). After powering the Central System on, the system is booted and NETOS is loaded by placing the NETOS disk (disk labeled: NETOS CENTRAL SYSTEM, NOVEMBER 1988, BOOTABLE SYSTEM) in disk drive 0. This disk is located in the drawer of the desk of System F. After a few seconds a menu will be displayed on the monitor. The selections are pretty much self-explanatory except for "B - Change Port Status". This prompt allows the operator to change the status of a port as either active or not active. Only active ports are polled for receiving packets.

Once the network is in operation, the monitor will display which host is being polled; when receiving a packet, which host the packet is being received from; and when transmitting a packet, which host the packet is being sent to. Also, displayed will be error messages if there were any errors.

Interfacing User Application Programs with the Network

The following files must be located in the default directory for the network to operate.

WARNING: THE EOF MARKER MUST OCCUR ON THE SAME LINE OF THE LAST DATA ENTRY OR AN ERROR WILL OCCUR READING THE DATA.

PTABLE.DAT - This file contains a listing of all host codes and the corresponding port id and baud rate associated with the host codes. Used by Layer 3.

HTABLE.DAT - This file contains a listing of all logical host name and the corresponding physical host code and process id associated with the logical host name. Used by Layer 5.

In order to interface with the lower ISO layers, the user application programs (Layer 7) must 'with' LAYER_6. The following files are required in order to link the user application program.

LAYER_1.JRL	LAYER_2.JRL	LAYER_5.JRL
LAYER_1.SRL	LAYER_2.SRL	LAYER_5.SRL
LAYER_1.SYM	LAYER_2.SYM	LAYER_5.SYM
SEND_2.JRL	LAYER_3.JRL	LAYER_6.JRL
SEND_2.SRL	LAYER_3.SRL	LAYER_6.SRL
SEND_2.SYM	LAYER_3.SYM	LAYER_6.SYM
RECV_2.JRL	LAYER_4.JRL	NETLIB.JRL
RECV_2.SRL	LAYER_4.SRL	NETLIB.SRL
RECV_2.SYM	LAYER_4.SYM	NETLIB.SYM

The user application programs typically provide a network service, where the network service consists of two programs; a server program and a user access program.

Before a network service can be provided, both the server program and the initiating program must initialize Layer 6. Then, the initiating program must send a request for service message to the corresponding server program. This is required in order to determine if the server program is online, ready, and capable of handling the request. Therefore, this request message must contain all necessary information to allow the service program to reach a decision as to accept the request or reject it.

The server program must be capable of sending a status message back to initiator. The status message indicates whether the request is accepted or rejected; and if rejected, the reason why. If accepted, the status message may contain further information for the initiator.

Once the appropriate handshaking of requesting a service and receiving an accept status message is complete, the initiator and network server both must be capable of transmitting and receiving messages back and forth as required.

Typically the initiator will be a user access program. However, there are special cases where the initiator will be a network server which is requesting service of another network server. For example, to print a list of files from

the Mass Storage System, this server must request service of the Printer Spooler System (HARTRUM, 1988).

Table IV lists the callable functions provided by Layer 6 to the user application programs. Tables V - VIII lists the current valid parameters. MSIZE defines the variable size of the message that can be communicated from point-to-point through the network. It is presently limited to a maximum of 510 by Layer 6. PRIORITY is currently limited between 0 - 9. INFO is defined as a string of five (5). INFO1 is defined as a string of 13. And, INFO2 is defined as a string of 5. Lastly, Table IX list all the status codes. Since there are no user application programs at the present time, these parameters have yet to be defined. Moreover, any changes to these will require changes in the Layer 6 software, since Layer 6 checks these input for validity. Also any changes to the SRCE/DEST parameters will require changes to the HTABLE.DAT file.

Table IV.
Layer 6 Callable Modules

```
procedure INITIALIZE_LAYER_6(STATUS : out Integer);

procedure SEND_NETOS_REQUEST(MSG_TYPE : in String;
                                 SRCE : in String;
                                 DEST : in String;
                             FILENAME : in String;
                             PRIORITY : in Character;
                          DEST_DEVICE : in Character;
                                 INFO : in String;
                               STATUS : out Integer);

procedure SEND_NETOS_STATUS(SRCE : in String;
                                 DEST : in String;
                     STATUS_CHARACTER : in Character;
                                INFO1 : in String;
                                INFO2 : in String;
                               STATUS : out Integer);

procedure SEND_NETOS_FILE(SRCE : in String;
                               DEST : in String;
                           FILENAME : in String;
                              MSIZE : in Integer;
                             STATUS : out Integer);

procedure SEND_NETOS_STRING(SRCE : in String;
                                 DEST : in String;
                          TEMP_STRING : in String;
                                MSIZE : in Integer;
                               STATUS : out Integer;
                           SYSTEM_MSG : in Boolean);

procedure RECEIVE_NETOS_REQUEST(MSG_TYPE : out String;
                                    SRCE : in out String;
                                    DEST : in out String;
                                FILENAME : out String;
                                PRIORITY : out Character;
                             DEST_DEVICE : out Character;
                                    INFO : out String;
                                  STATUS : out Integer);

procedure RECEIVE_NETOS_STATUS(SRCE : in out String;
                                    DEST : in out String;
                        STATUS_CHARACTER : out Character;
                                   INFO1 : out String;
                                   INFO2 : out String;
                                  STATUS : out Integer);
```

```
procedure RECEIVE_NETOS_FILE(SRCE : in out String;
                             DEST : in out String;
                         FILENAME : in String;
                           STATUS : out Integer);

procedure RECEIVE_NETOS_STRING(SRCE : in out String;
                               DEST : in out String;
                        TEMP_STRING : out String;
                             STATUS : out Integer);
```

Table V.
MSG_TYPE Parameters

"SPO"	-	Spooler Request
"MSS"	-	MSS Request
"QRY"	-	DBMS Request

Table VI.
SRCE/DEST Parameters

"SPOOLER "	-	Spooler Printer System
"MASS-STORE"	-	MSS Storage System
"DATABASE "	-	Database System
"REMOTE-A "	-	Remote System 'A'
"REMOTE-C "	-	Remote System 'C'
"REMOTE-D "	-	Remote System 'D'
"REMOTE-E "	-	Remote System 'E'
"REMOTE-F "	-	Remote System 'F'
"REMOTE-G "	-	Remote System 'G'
"REMOTE-H "	-	Remote System 'H'
"REMOTE-I "	-	Remote System 'I'
"REMOTE-J "	-	Remote System 'J'
"**********"	-	Any System

Table VII.
DEST_DEVICE Parameters

'A'	-	Alps Printer
'*'	-	Any Printer (don't care)

Table VIII.
STATUS_CHARACTER Parameters

'A'	-	Test_A status
'B'	-	Test_B status
'C'	-	Test_C status
'D'	-	Test_D status
'E'	-	Test_E status

Table IX.
Status Codes

0 - No errors
101 - Invalid baud rate
102 - Error initializing the port
201 - Timeout sending transmit request 'B'
202 - Timeout waiting transmit acknowledge 'A'
203 - Non-transmit acknowledge received
204 - Timeout waiting ACK or NAK
205 - NAK received
206 - Non ack/nak received
211 - Timeout waiting to receive a transmit request
212 - Non-transmit request received
213 - Timeout waiting to send transmit acknowledge
214 - Timeout waiting to get 1st 4 bytes of a frame
215 - The 1st byte of the frame was not an ASCII.SOH
216 - Timeout getting the rest of the frame
301 - SRC/DST code not found in table
302 - Invalid SRC code received
303 - Invalid DST code received
401 - One end of the link is already in use
402 - All channels are already in use
403 - Invalid channel connection response
404 - Invalid channel request received
405 - Channel not found while trying to close
406 - Channel not found while trying to send
501 - SRCE/DEST names not found
502 - SRC/DST and SPN/DPN codes not found
601 - Not a valid msg_type
602 - Not a valid srce/dest
603 - Filename does not begin with alpha-character
604 - Not a valid priority
605 - Not a valid DEST_DEVICE
606 - Not a valid STATUS_CHARACTER
607 - Not a valid MSIZE
608 - MSG_TYPE not a NETOS status message

Bibliography

Booch, Grady, Software Engineering with Ada. Menlo Park, California: Benjamin/Cummings Publishing Company, Inc., 1986, pp xiii-8.

Hartrum, Thomas C., "NETOS - Network Operating System at AFIT," Department of Electrical and Computer Engineering, School of Engineering, Air Force Institute of Technology, WPAFB, Dayton, OH, Feb 1988.

Tanenbaum, Andrew S., Computer Networks. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1981, pp 15-21.

Zimmerman, Hubert, "OSI Reference Model - The ISO Model of Architecture of Open Systems Communications," IEEE Transactions on Communications, Vol. COM-28(4), April 1980, pp 425-432.

VITA

Robert Rodriguez [redacted] He entered active duty in the United States Air Force in September 1977 and attended the Weapons Control System (F4E) training at Lowry Air Force Base, CO. Upon completion, he was assigned to Homestead Air Force Base, FL where he served as a Weapons Control System (F4E) Mechanic. He received an honorable discharge in December 1980. He immediately attended the University of Florida. He graduated in 1983 with a Bachelor of Science Degree in Electrical Engineering. Upon graduation, he received a commission in the USAF by attending the Officer Training School at Lackland Air Force Base, Texas. He was assigned to Tyndall Air Force Base, FL where he held the position of Chief, Facilities Section. He entered the School of Engineering, Air Force Institute of Technology, in June of 1987.

[redacted]

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release, distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/88D-43	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFIT 8b. OFFICE SYMBOL (If applicable) ENG	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
See Box 19 (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
Robert Rodriguez, B.S., Capt, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1988 December	106

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	computers, computer programming, microcomputers, Communication networks
09	02		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: Implementation of the NETOS Operating System in Ada with Modifications to Allow Variable Length Messages.

Thesis Chairman: Bruce L. George, Capt, USAF

This thesis redeveloped the Network Operating System (NETOS) software, which is patterned after the OSI 7-Layer Model and runs on a Local Area Network at the Air Force Institute of Technology, from the programming language C to the programming language Ada, with modifications to support variable length messages.

The approach taken used a software development methodology which contains the following phases; requirements analysis, design, implementation, and testing. The requirement analysis phase consisted

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☑ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Bruce L. George, Capt, USAF	22b. TELEPHONE (Include Area Code) 513-255-3576 22c. OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Block 19 cont.

of an enumerated listing of the requirement specifications supported by SADT diagrams. The design phase transformed these diagrams to a structural chart representation of the design. Implementation of the software was written in Janus/Ada for the work stations and Whitesmith C for the central system. Testing was an integral part of the implementation phase and was accomplished at each level of the 7-Layer model.